Introduction to Programming with Python

Riding the Serpent

Anshul Nigham & Rob Tirrell

November 12, 2010

About Us

Anshul Nigham

- Began programming at 12 on on 8086 PC-XT (a full-fledged computer 20x slower than an iPhone).
- Converts caffeine into code for a living at Yahoo.

Rob Tirrell

- In the second year of a five to six year sentence in the Butte lab (http://buttelab.stanford.edu/), an entirely 'dry' lab (computers only – the only other equipment necessary is a coffee machine).
- First language was Python, spends most days writing code in R, Python, Ruby and C++.

What is Python?

- First released in 1991 by a Dutchman named Guido van Rossum (GvR).
- That's Self-Appointed Benevolent Dictator for Life (SABDFL) van Rossum to the rest of us.
- An interpreted, high-level language with flexible typing.
- Currently on its third major release... in other words, it's been around the block and has withstood the test of time.

A Satisfied User

Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language.

– Peter Norvig, Director of Search Quality at Google
and Computer Science Superstar

Other Satisfied Users

- **AstraZeneca** uses Python in drug discovery pipelines.
- **Phillips'** fabrication plants are managed in Python.
- **Industrial Light & Magic** (Star Wars) employs Python for process management.
- At **Yahoo**, **Anshul** developed an AdWords optimizer in Python.
- In **Rob's** work, people use Python at every point in the research pipeline (preprocessing and sanitization, standard analyses, data aggregation and integration, and so forth).
- It may be new to you, but according to TIOBE's programming languages index, Python is the sixth-most popular in the world.
- A list of anecdotes can't quite prove a point, so we'll try to justify why you should care about and use Python.

Good for Them, What's in it for You?

Power

- Python facilitates rapid development. It comes with a huge collection of software (libraries) for many purposes.
- There is a sizeable, vibrant and very helpful Python community/ecosystem, should you run into trouble or seek advice.

Clarity

- Python is remarkably clear and readable compared to many other languages.
- It actually takes some effort to write difficult-to-understand programs.

Before we can reveal Python, let's consider the high-level **process and principles of programming** (trust us, we know they're abstract and seemingly useless, but it's critical to really get what they mean).

In Case of Emergency, Panic!

- But, really, please ask us questions – raise your hand or shout it out.
- As this is an introductory course,

Principal Principles of Programming (1)

Computer programming *or* ***coding*** *is the process of designing, writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. This source code is written in a programming language. The purpose of programming is to create a program that exhibits a certain desired behaviour. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.*

– Wikipedia

Principal Principles of Programming (2)

~~***Computer programming*** *or* ***coding*** *is the process of designing, writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. This source code is written in a programming language. The purpose of programming is to create a program that exhibits a certain desired behaviour. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.*~~

The Heart of the Matter

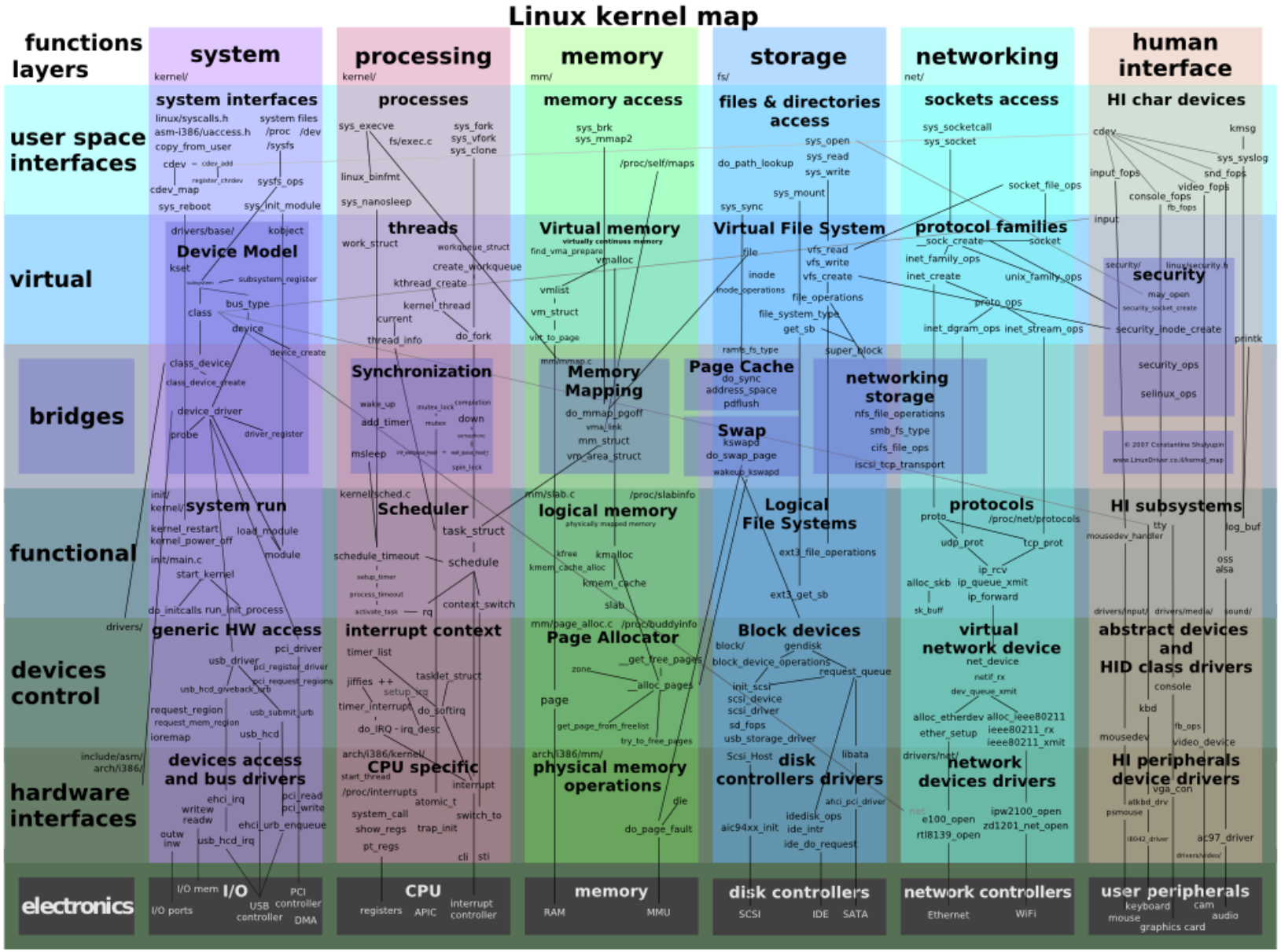
- It's definitions like this that turn people off to programming.
- At its core, programming is just writing a set of instructions which a computer runs on your behalf.
- How you conceive of and design those instructions is the fun part (and the challenge).

Principal Principles of Programming (3)

Decomposition

- **Decomposition** is the means by which a complex problem or system is broken down into parts that are easier to understand, program and maintain.
- This is one of the fundamental principles of design and programming: we decompose the logical structure into smaller, reusable units which we build one-at-a-time.
- Sometimes, how this decomposition should proceed isn't immediately clear. Then your first step should be thinking in depth about the problem you're facing, and in what ways it can be reduced into component subproblems.

Principal Principles of Programming (4)



– Wikipedia

Principal Principles of Programming (5)

Divide and Conquer

- Imagine you are a pin manufacturer in 18th century England. To achieve maximum efficiency, you will make pins step-by-step: pounding the metal into a sheet, cutting the sheet into small strips, heating the protopins, elongating the pins, punching an eyelet, polishing the pins, etc..
- This is simple enough: just a stepwise series of transformations. More often, a computer program will be more complex, and have multiple interdependent parts.

Principal Principles of Programming (6)

Thinking about a Twitter iPhone App

- Let's motivate this with an example.
- Break into groups with your nearest neighbors and discuss how you would design this application.
- What are the application and user operations we would want to support, and what are their requirements?

The Interpreter (1)

>>>

- As Python is an interpreted language (the computer reads, understands (compiles) and executes it on the fly, instead of reading and compiling ahead of time (as with C, C++, and many others).
- Because of this, we can use Python interactively, which is extremely helpful for designing and troubleshooting code.

IDLE

- IDLE is Python's interpreter shell, which we'll use to walk through examples throughout the course.
- For those of you with Macs, you can open a Terminal and launch Python by typing `python`.

The Interpreter (2)

The `print` Function

- Strings are surrounded by single (') or double (") quotes, which are equivalent (but you must use the same type for any particular string). e.g., if you want a string to have a contraction `"mustn't"`, you can use double quotes to surround it, like `"The white whale mustn't breach, Moby is waiting."`.
- `print` is a core Python function, which by default outputs text to the screen. It's somewhat special, in that parentheses are optional when using (calling) it.

A Longstanding Tradition

Three equivalent `'Hello, World!'`s:

```
print 'Hello, World!'
print "Hello, World!"
print('Hello, World!')
```

The Interpreter (3)

Not Your Daddy's Desktop Calculator

- Among other uses, we can employ Python as a calculator.
- At the interpreter (`>>>`), type `4 + 2`. You should see `6`. So far, so good – naturally, other operators are available (e.g. `+ - * /`).
- A player's **slugging percentage** (in baseball) is calculated as `(singles + 2 * doubles + 3 * triples + 4 * homers) / at bats`.
- In `1920`, Babe Ruth appeared at the plate `458` times, cranking out `73` singles, `36` doubles, `9` triples and `54` home runs. What was his slugging percentage?[1]

[1] The divisor needs to be a floating-point number (`458.0`), otherwise Python will drop the fractional component of a number. This behavior can be a pain, and has been changed in Python 3.

The Interpreter (4)

More than a Glorified Calculator – Saving it for Later

- It can be tedious to have to type out that expression every time. Luckily for us and our fingers, we can store the results of calculations in **variables** very simply in Python.
- For example, `sp = (73 + 2 * 36 + 3 * 9 + 4 * 54) / 458.0`
  If you mess up the numbers, you can always try again. The new value will replace the old one.
- Voilà! Try `print`ing the result.

The Interpreter (5)

Back to the Babe

- Another useful stat is `on-base percentage` (OBP), calculated as `(H + BB + HBP) / (AB + BB + HBP + SF)`. In English, `(hits + walks + hit by pitches) / (hits + walks + hit by pitches + sacrifice flies)`.
- In 1920, the Babe had `172` hits, walked `150` times, was hit by `3` pitches, and had no sacrifice flies (and `458` at bats). What was his OBP (hint: store this in a variable `obp`)?
- Finally, to close out the baseball, **on-base plus slugging** (OPS) is a concise statistic frequently used to quantify players' offensive contributions. Since we have both of these measures already stored, calculating Ruth's OPS is as simple as `sp + obp`!
- Hopefully this illustrates the value of saving the results of intermediate computation. You'll thank yourself later.

Datatypes (1)

What are They?

- A **datatype** refers to a location in the computer's memory and the type of information stored there.
- Numbers can be of the `int` (integer) datatype, like `4`, or the `float` (floating point) datatype, like `4.0`).
- Text uses the `string` datatype, like `'Four score and seven years ago...'`.
- True and false values are `bool` (boolean) datatypes, in Python these are `True` and `False`. There is also a special value called `None`, that indicates 'no value'.
- We can view the type of any variable using `type`, so typing `type('bananagram')` returns `<type 'string'>`.

Datatypes (2)

More Advanced Datatypes

- Obviously, more complex programs require more complex datatypes (depending on the complexity, they may be referred to as 'data structures').
- The two most important in Python are the `list` and `dict` (dictionary).
- A list:

```
some_list = [12, 'monkeys']
```

- A dictionary:

```
some_dict = {
  'New York': 'A state in the US.',
  42: 'A completely irrelevant number.',
  'today_is_sunday': False
}
```

Datatypes (3)

Duck Typing

- The Python philosophy: if it looks like a duck, and it quacks like a duck, then it's a duck.
- In other words, variables are not restricted to contain a particular datatype. We can re-assign them to anything we want:

```
var = 44
var = 'The Life of Brian'
var = [
  'Episode IV: A New Hope',
  'Episode V: The Empire Strikes Back',
  'Episode VI: Return of the Jedi'
]
var = {
  'aardvark': 'a nocturnal burrower from Africa',
  'antbear': 'another name for an aardvark'
}
```

- Functions exist to coerce strings like `'1024'` into the corresponding floating-point or integer numbers (c.f. `int()` and `float()`), or numbers into the corresponding string (c.f. `str()`), and so on.

Datatypes (4)

Rolling Our Own Classes

- We can also write custom classes with the `class` keyword, with which we can define datatypes (or data structures) that store arbitrary and have a set of methods to manipulate that data.

```
class MyTestClass:
  def __init__(self, name):
    self.name = name
  def say_hello(self, other_name):
    print self.name + ' says hi to ' + other_name
my_test_class_instance = MyTestClass('Rocko')
my_test_class_instance.say_hello('Clarissa')
# >>> 'Rocko says hi to Clarissa'
```

- The ability to extend the language with our own classes is extremely useful, but for concerns of time we'll just dangle the idea in front of you and leave it that. Any book on Python will have a more thorough treatment of classes.

Introduction to Functions and Methods (1)

Making Things Happen

- **Functions** and **methods**[2] are procedures that work on variables to transform or otherwise alter them.
- Change a string to all uppercase:

```
some_string = 'julius'
some_string = some_string.upper()
some_string
# >>> 'JULIUS'
```

- Access an entry (sometimes called an element) of a dictionary:

```
some_dict[42]
# >>> 'A completely irrelevant number'
```

[2] There is a difference: methods are attached to and operate on their objects, while functions stand alone. e.g., `say_hello` in the class on the preceding page was a `method` associated with `MyTestClass` objects. Don't worry too much about this now, we'll come back to it later (and in this course will only write functions, anyway).

Introduction to Functions and Methods (2)

A Few More Examples

- Many of Python's datatypes support this sort of access (called indexing).

```
some_string[1]
# >>> 'U'
```

- This is an important point: in Python, the first element of a collection is the `[0]`th" one, available at `collection[0]`.
- Similarly, for lists:

```
some_list[1]
# >>> 'monkey'
```

Brief Interlude: Setting Up KomodoEdit

New Release!
ActiveState
Komodo® Edit 6

KomodoEdit should already be installed on the provided computers, if not, visit http://www.activestate.com/komodo-edit/downloads to grab it.

Functions (1)

Writing Your Own with `def`

- The `def` keyword is used to create new functions.
- Functions always `return` a value. Functions which do not explicitly `return` a value (have no `return` statement) implicitly `return None`.

```
# This function has no parameters and...
def test_func_1():
  print "I'm useless!"
```

Functions (2)

```
# This function has no parameters and returns None.
def test_func_2():
  pass

# This function has one parameter - a number.
# It adds 10 and returns that value.
def add_10(parameter_1):
  return parameter_1 + 10
```

Write Your Own

Try writing a simple function that takes one parameter (a string) and uses + to display 'Hello <your string goes here>'.

Igpay Atinlay (1)

Pig Latin

Pig latin is a nonsense language, that can be generated easily for any English word, by moving the word's first character to the end and adding `'ay'`. Then, `pig_latinize('pork') = 'orkpay'`

String Indexing and Slicing

- Recall that we can access single elements of a collection using `[]`. We can also access longer parts of the string (which is a collection of characters) using **slices**.

```
green_eggs = 'green eggs and ham'
print green_eggs[6:10]
>>> 'eggs'
```

- Slices range from `0`, the first element, to `-1`, the last element.
- Omitting the start or end defaults to 'from the beginning' or 'to the end', respectively. Thus, `some_collection[:] == some_collection`.

Igpay Atinlay (2)

Reading User Input with `raw_input`

- Often, we'll want to get input from a user (e.g., a name).
- The `raw_input` function does exactly that. We pass it one argument, a string that will be printed as the prompt (e.g. `'Enter your name:  '`).
- It `returns` whatever the user enters in response[3]:

```
favorite_color = \
  raw_input('What is your favourite color? ')
```

Then `favorite_color` will be `'red'`, or `'Black'`, etc..

[3] The `'\'` just allows us to continue a Python expression on the next line, so it doesn't run off of the slide :).

Igpay Atinlay (3)

Make Your Own Nonsense

- Using slicing, write a function that takes one parameter (a string) and `returns` the pig latinized version of the string.
- We'll have to make one slice, access one element of the string directly, and then add 'ay' at the end: `<all but the first character> + <the first character> + 'ay'.`

Igpay Atinlay versus igPay atinLay (4)

Improving Pig Latinization

You probably noticed that capitalized characters look out of place in the first version of the function. There are two string methods, `lower` and `capitalize`[4], that we can use to fix this problem.

- Modify the function so that it converts the pig latinized string to lowercase, then capitalizes it.
- We can do all of this in one line, but it is easier to read and understand if you store the intermediate pig latinized string in its own variables, like `pig_latinized_string`.
- Helpfully, we can 'chain' method calls, in other words, `some_string.lower().capitalize()` works!

[4] An exhaustive list of string methods is available at http://docs.python.org/library/string.html. All of the builtin libraries are well-documented, see http://docs.python.org

Dealing with Uncertainty (1)

Truth Testing

- Few programs are just a straight pipeline of unbiased transformations to data – most of the time, we'll want to branch out and make decisions based on input.
- Python has the `if`, `elif` and `else` keywords that provide this ability.

```
some_boolean = true
if some_boolean:
  print 'Truly today is a special day!'
else:
  print 'Back to the grind, peasant.'
```

Dealing with Uncertainty (2)

Equality

- We can also use ==, $<$, $<$=, and so on.
- The instructions following an `elif` keyword are only triggered if no previous conditions are met and the `elif` condition is.

```
strength = 15
if strength >= 15:
  print 'You pull the sword from the stone.'
  print 'All hail King Arthur!'
elif strength >= 10:
  print 'Solid effort.'
  print 'But you were born to work in the mines.'
else:
  print 'Nope! Not even close.'
```

Looks like it's deep, dark and dirty for this guy.

Case Study: The Candy Shoppe (1)

A Profitable Business Proposition

- You run a candy shoppe, selling all variety of popsticks, licorice screamers, fireballs and the like.
- Little kids are big business you, and these little sugar fiends are spending their parents money and are not very sensitive to changes in price. Being a businessperson first and foremost concerned with the bottom line, you want to charge anyone less than `14` years old a little bit extra (`25`% over regular price).
- But you're not a monster, and if the customer's blood sugar is below `2.5`, you'll give him or her a `15`% discount (with no additional penalty for being less than `14`).
- You want this to be done transparently, so you will add a function to your cash machine that jacks up the price depending on the customer's information (which is stored in the customer's loyalty card).

Case Study: The Candy Shoppe (2)

Customer Information

- Each customer's loyalty card has several attributes encoded in the magstripe.
- These attributes are loaded into the cash machine when you scan the card, and are available to the software as a dictionary.

```
customer_info = {
  'name': 'Rob Tirrell',
  'email': 'rpt@stanford.edu',
  'age': 24,
  'blood_sugar': 5.0, # mmol / L
  'likes_dark_chocolate': False,
  'astrological_sign': 'Gemini'
}
```

Case Study: The Candy Shoppe (3)

Implementation

- Recall that we access dictionary values by their key, so for the above example, `customer_info['age']` will `return` 24.
- As we might except, we can make comparisons with dictionary values directly, without storing the value in a separate variable:

```
if customer_info['age'] > 65:
  print 'Senior citizen discount!'
```

- We want to write a function, that given the purchase price and a customer's information, 'adjusts' the price appropriately. It should `return` a number and look something like:

```
def compute_price(base_price, customer_info):
  pass # Fill in here!
```

Importing and Using Modules (1)

- So far, we've only played with the core of Python, so let's dive into the standard library (the suite of useful software that comes with Python).
- To load any one of these libraries, we use `import` statements:

```
import random
# Generate an integer from 1 to 6 (inclusive).
random.randint(1, 6)
# Generate an integer from 1 to *5* (inclusive).
random.randrange(1, 6)
```

Importing and Using Modules (2)

Who's Feeling Lucky?!

- Let's use random to pick the winner of a lottery. First, define the players:

```
lottery_players = [
    'Rufus',
    'Sun',
    'Tania',
    'Boris',
    'Rocky',
    'Daisy'
]
```

- Then, picking a winner with `random` is stupid simple:

```
print random.choice(lottery_players)
```

Importing and Using Modules (3)

Shufflin'

- How about one more quick example with the same people as before. We want to split the people into two teams, so we can sell tickets to a 3v3 wrestling extravaganza to pay off their gambling debts.
- Again, ask and `random` provides the `shuffle` function, which randomizes the ordering of a list **in-place** (that is to say, it `returns None`).

Importing and Using Modules (4)

Picking Sides

- Let's shuffle the list, and then save the number of players in a variable `number_of_players` using the builtin `len` function.[5].

```
number_of_players = len(lottery_players)
print 'Team 1:'
print lottery_players[:number_of_players / 2]
print 'Team 2:'
print lottery_players[number_of_players / 2:]
```

[5] `len(object)` is really just a proxy for the method call `object.__len__()`. This is also the case for some of the operators we've seen, e.g., `object[i]`, which just calls `object.__getitem__(i)` behind the scenes, and `object + 1` becomes `object.__add__(1)`.

Choose Your Own Adventure

We have a couple of choices for what to do next. We can:

- Write a program that analyzes word usage in a document, featuring a brief introduction to natural-language processing (NLP) theory and techniques.
- Write a program that uses Google web services to 'fight' terms against each other (in the spirit of Google Fight).
- Mangling Mark Twain: code a 'random writer' that reads a document and generates a somewhat sensible-sounding babble based on it.

  > *By jingo! that reminds me of a droll dog of a thieving line– the high standard held up to the mode in which her mouth and brow there was something alien and ill-understood the impetuosity. His reason for hastening it–if he scrupulous explorer to be saluted with those cheerful view of all the plans, and took lasting impression of Celia, Tantripp, stooping and getting a bit of a note saying you don't see anything of that sort of challenge me.*
  >
  > – Generated from Middlemarch by Mary Ann Evans

So We've Sold You on Python, What's Next? (1)

- There are **tons** of freely-available resources on the internet to continuing experimenting and learning. The best place to start is probably Dive Into Python 3 (http://diveintopython3.org) by Mark Pilgrim. As the name indicates, it covers Python 3 (we are using 2.7, some things differ but most of what we've gone over is the same).
- We haven't even nearly covered the standard library (the stuff that comes with every version of Python). Honestly, I haven't used or even heard of most of it.
- Python has builtin modules for scraping from the internet (`HTMLParser`), (extremely advanced) string searching and manipulation (the strange world of regular expressions, in the `re` module), creating, deleting and managing files and directories (`os` and `shutil`), creating and extracting compressed files (`gzip`, `bz2`, `zipfile`, `tarfile`), interacting with FTP servers (`ftplib`), and on and on and on $\rightarrow \infty$.

So We've Sold You on Python, What's Next? (2)

Stepping away from builtin functionality, there are many other mature libraries in the wild:

- If you want to write websites, check out Google's App Engine (http://code.google.com/appengine/). It provides free hosting services and holds your hand along the way (that's a good thing!). For more advanced websites, look into the Django Project (http://www.djangoproject.com/) (App Engine is actually based on Django).
- There are libraries for image manipulation (the Python Imaging Library, http://www.pythonware.com/products/pil/), numerical and scientific calculation (NumPy/SciPy, http://numpy.scipy.org/), bioinformatics (BioPython, http://biopython.org/wiki/Biopython), plus a whole slew of others for most every purpose imaginable.

... And a Universe of Other Languages

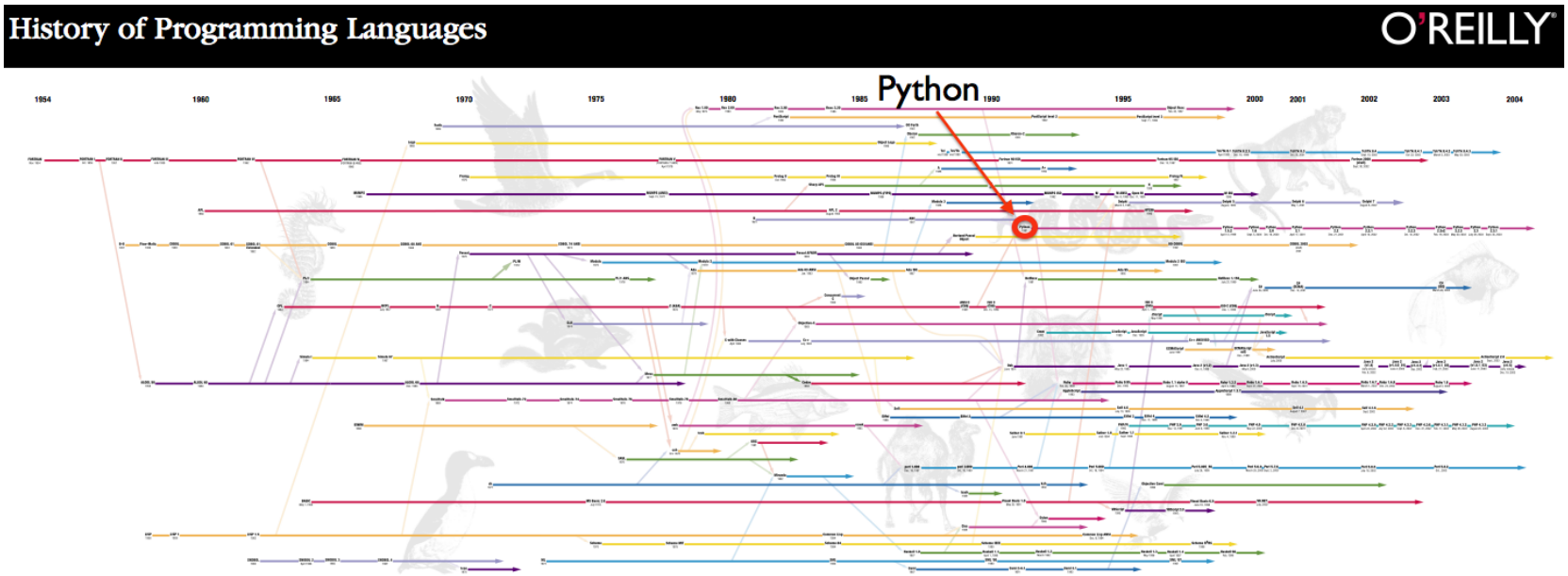


Image credit: O'Reilly (http://oreilly.com/news/graphics/prog_lang_poster.pdf)

Thanks!

- We hope this little taste of the joys of programming whetted your appetite – if so, this is the best place in the world to be.
- If you have any questions about Python, programming, or computer science in general, feel free to email either of us.
- Anshul: anshul@gmail.com, Rob: rpt@stanford.edu.
- `print 'Farewell, and godspeed!'`